AD-A254 894

A Genetic Algorithm for Search Route Planning

By

J.R. Van Zandt

July 1992

Prepared for

Program Manager
Combat C³ Systems
Electronic Systems Division
Air Force Systems Command
United States Air Force

Hanscom Air Force Base, Massachusetts

DTIC
ELECTE
SEP 02 1992
S **A** **D**

92-24214

235050

49P⁵

92 9 01 014

REVIEW AND APPROVAL

This technical report has been reviewed and is approved for publication.

JANET M. POLANECZKY, CAPT, USAF
Project Manager

FOR THE COMMANDER

ROBERT S. MCCULLOCH, COL, USAF
Program Manager, Combat C^3 Systems

REPORT DOCUMENTATION PAGE

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1992	Final

4. TITLE AND SUBTITLE

A Genetic Algorithm for Search Route Planning

5. FUNDING NUMBERS

F19628-89-C-0001
60260

6. AUTHOR(S)

Van Zandt, James R.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The MITRE Corporation
202 Burlington Road
Bedford, MA 01730

8. PERFORMING ORGANIZATION REPORT NUMBER

MTR-11177

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Program Manager, Combat C^3 Systems   (ESD/SZ)
Electronic Systems Division, AFSC
Hanscom AFB, MA 01731-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ESD-TR-92-262

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Planning an airborne search for a relocatable target involves minimizing the risk to the crew while maximizing the estimated likelihood of finding targets. When the number of potential sites is small, the problem reduces to finding the best (usually the shortest) route connecting them. When the number of sites is large, however, the planner must also decide which of them are to be visited. This report presents a genetic algorithm for selecting both an appropriate set of sites to visit and an appropriate routing pattern. The procedure also accounts for the finite turning radius of the aircraft.

14. SUBJECT TERMS

Constrained Optimization
Covering Salesman Problem
Genetic Algorithm
Routing
Spacefilling Curve
Traveling Salesman Problem

15. NUMBER OF PAGES

51

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

ACKNOWLEDGMENTS

Accesion For		
NTIS CRA&I	☑	
DTIC TAB	☐	
Unannounced	☐	
Justification		
By		
Dist.ibution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TABLE   **PAGE**

SECTION 1

INTRODUCTION

When a strategic bomber must search for a relocatable target among a number of sites, the mission planning must include choosing a search route. This routing problem is closely related to the classical traveling salesman problem (TSP). However, the bomber need not overfly each site, but must only come within sensor range of it. An overflight of one site may provide adequate sensor coverage of nearby sites as well. Thus, the search route planning problem is better modeled by the *covering salesman problem* (CSP) formulated by Current and Schilling [1]. The CSP is a generalization of the traveling salesman problem in which the tour need not visit every city, but every city must be within a certain distance of some city on the tour. A previous report presented a heuristic based on spacefilling curves for this problem [2]. The result is a two-dimensional route which passes within a specified distance of all the sites of interest, while approximately minimizing the total distance traveled.

When the number of potential sites is large, the bomber's time and fuel constraints may preclude visiting all of them. The planner must then decide which of them are to be visited. This report extends the above procedure to include the selection of an appropriate set of sites. The new procedure also evaluates alternative routing patterns and accounts for the finite turning radius of the aircraft.

Selection of the sites to visit is handled by imbedding the heuristic router within an optimization program. The outer program selects an appropriate set of sites to be visited. The heuristic router finds a search route covering those sites. The outer program computes a score for the route. It then adjusts the sites to be visited in an attempt to optimize the score (while satisfying time and fuel constraints) and repeats. For this combinatorial optimization problem, a *genetic algorithm* is used.

Section 2 shows how a flight path which accounts for the finite turning radius of the bomber can be constructed. Section 3 describes extensions of the fast routing algorithm to include the "strips" and "arbitrary insertion" heuristics. Section 4 describes the genetic algorithm and its application to the search routing problem. Section 5 displays typical results. A concluding discussion appears in section 6.

1

SECTION 2

FLIGHT PATHS

A flight plan includes a set of *waypoints* which a plane flies through in succession. In this work, waypoints are two-dimensional. When the waypoints are well separated, the flight path can be approximated by straight lines between the waypoints. However, for closely spaced waypoints, high aircraft speeds, and/or restricted aircraft maneuverability, the finite turn radius of the aircraft must be taken into account.

By convention, a flight path consists of straight lines and circular arcs. One planning objective might be to find the shortest such path whic'. passed through the specified sites. It might be constructed using the following manual procedure: Start with a flat map of the flight area. Insert a pin at each of the points to be visited. Drop over each pin a hollow cylinder of a size corresponding to the desired turn circle. Thread a string around the cylinders in the chosen order and draw it tight, forcing each cylinder against its pin. (We assume the cylinders do not interfere with each other.) The string then follows the shortest path for the chosen visitation order.

This manual procedure generates a flight path which visits each waypoint halfway through a turn. This doubles the number of significant locations the navigator must keep track of (search sites plus turn points). It requires accurate flying, since an incorrect airspeed or bank angle will lead to an incorrect turn radius and possibly a missed search site. It also means that the aircraft will never be level when overflying the site. For these reasons, as well as convenience, we have instead implemented the simpler *waypoint turn algorithm*: The aircraft initiates each turn as it passes a waypoint. The aircraft then flies along a *turn circle*, with its center at a distance R from the the waypoint and at right angles from the original flight path. Ordinarily, the aircraft turns toward the next waypoint, as shown in figure 1. However, if this would put the next waypoint within the turn circle, the turn can be made in the opposite direction instead, thus adding a *loop*. This is shown in figure 2. These two cases suffice to handle arbitrarily difficult routing problems, such as the closely spaced waypoints shown in figure 3.

Appendix A presents some mathematical tools which were used to automate this procedure.
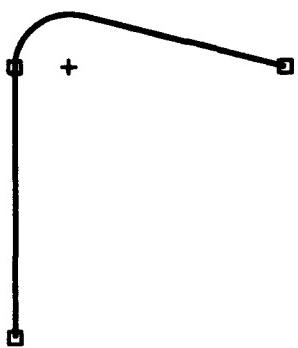
Figure 1. Aircraft Turns Toward Next Waypoint
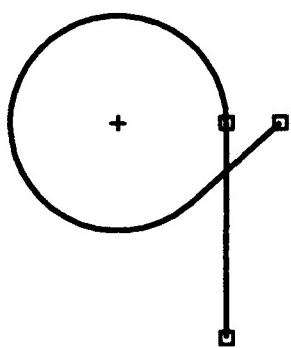
Figure 2. Aircraft Turns Away from Next Waypoint

4
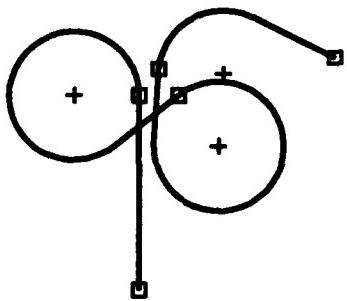
Figure 3. Waypoint Turn Algorithm Applied to Closely Spaced Waypoints

SECTION 3

ROUTING PATTERNS

A previous report [2] presented an algorithm based on spacefilling curves for finding heuristic solutions to the traveling salesman problem (TSP). This section describes two other TSP heuristics which can also be adapted to the search routing problem.

The spacefilling curve is a continuous function from the unit interval onto the unit square. The TSP heuristic required a function $f(x, y)$, a pseudoinverse of the spacefilling curve, which was used to order the cities to be visited. Different spacefilling curves lead to different functions. The function for the Hilbert curve is shown in figure 4. This function preserves *nearness*, in that only nearby points in the plane are given similar values.

The "strips" method described by Karp and Steele [3] is an older heuristic for the planar TSP. For N cities, it consists of the following:

STRIPS HEURISTIC.

    (i)   *Divide the area into $\sqrt{N}$ horizontal strips.*

    (ii)   *Sort the cities along each strip*

    (iii)   *Visit the cities in the first strip from right to left, then those in the second strip from left to right, etc.*

The "strips" heuristic can be implemented in the same way as the spacefilling curve heuristics, by defining an ordering function similar to that shown in figure 5 (for three strips). For large numbers of sites distributed uniformly in the unit square, the strips heuristic generates TSP solutions about 22 percent longer than optimal (*vs.* 25 percent over optimal for the spacefilling curve heuristic using Sieripinski's curve). The strips heuristic is more sensitive to local variations in site density.

The "arbitrary insertion" heuristic for the TSP described by Golden [4] has also been adapted to the search routing problem as follows:

ARBITRARY INSERTION HEURISTIC.

    (i)   *Start with a path directly from the start to the finish.*

    (ii)   *Repeat until all cities have been added:*
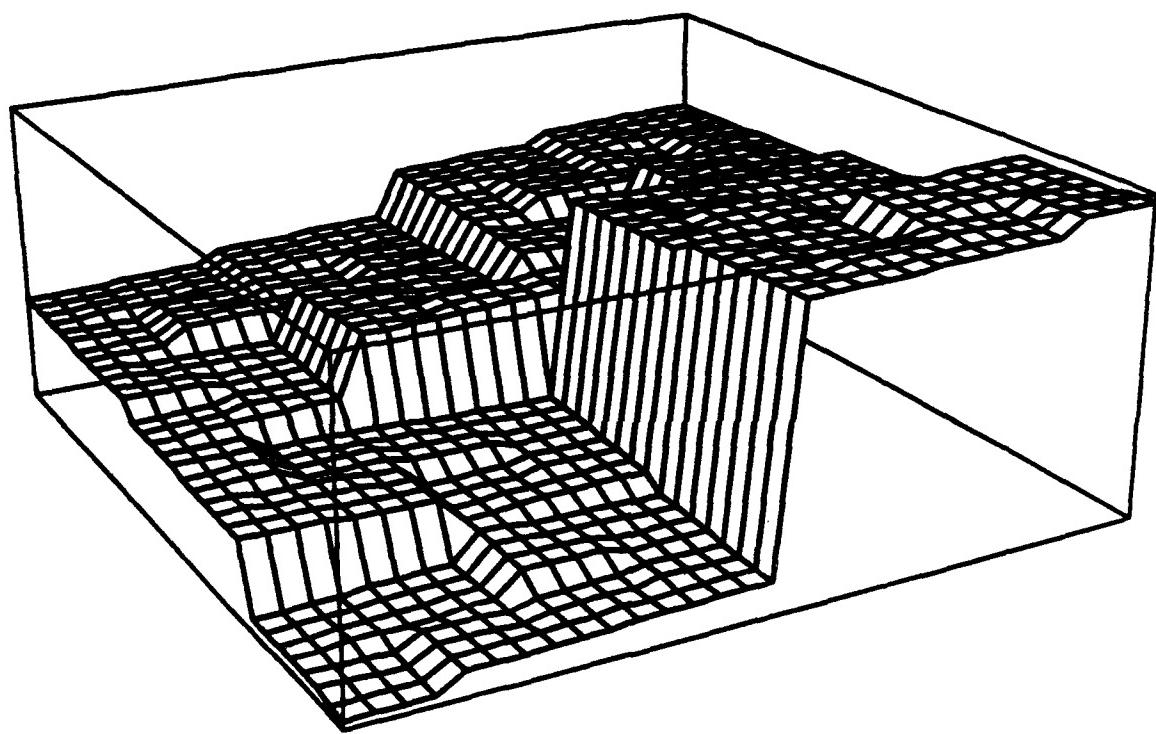
Figure 4. Ordering Function for Hilbert Curve
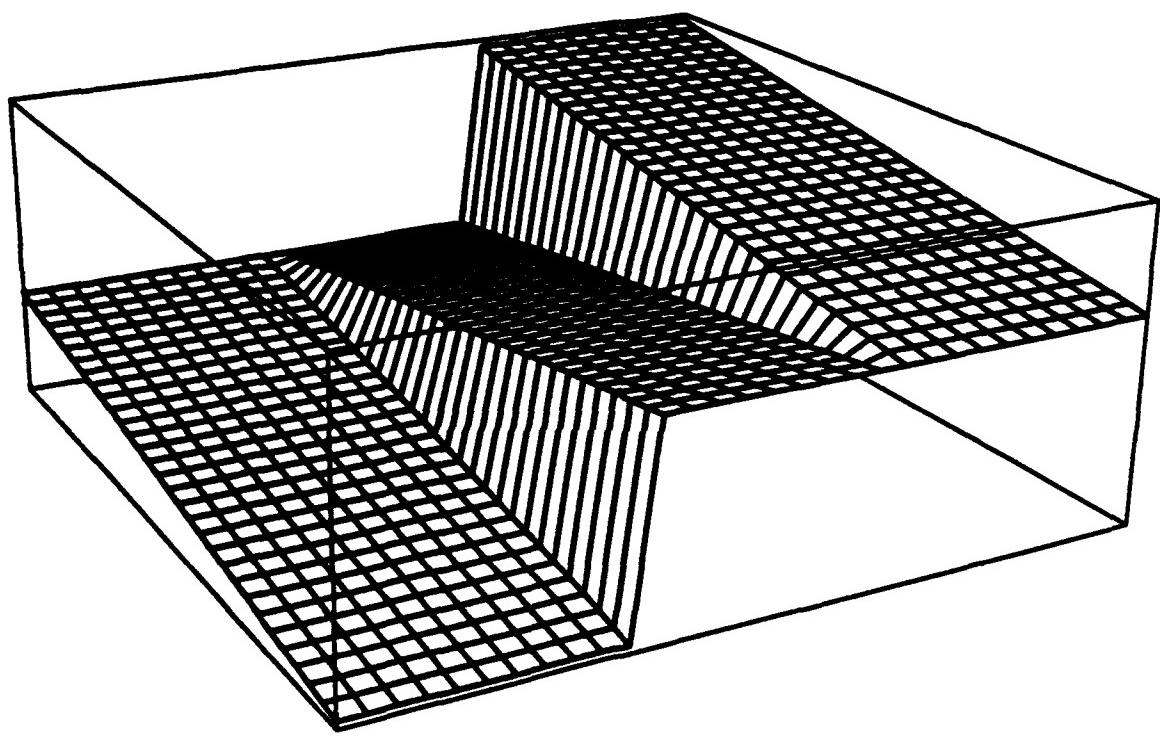
Figure 5. Ordering Function for Strips Heuristic

(a) Arbitrarily select city k not in the path to enter the path

(b) Find the edge (i, j) in the path which minimizes the cost of inserting city k and insert it there.

The arbitrary insertion heuristic requires $O(N^2)$ steps, while the spacefilling and strips heuristics require only $O(N \ln N)$ steps.

The arbitrary insertion method could be used in a semiautomated router in which an operator specifies part of the route (initial and final points, and some of the intermediate sites) and the computer adds the remaining sites. The initial route could include multiple visits to some sites.

The spacefilling and strips heuristics require a preprocessing step: the city locations must be rotated and scaled into the unit square. This is not necessary for the arbitrary insertion heuristic.

SECTION 4
GENETIC ALGORITHMS

The traditional optimization procedures are calculus-based methods (solve for locations where the gradient vanishes, or *hill climbing*: move in the direction of the local gradient), random searches (generate points at random and keep the best), and enumeration (generate all possible points and keep the best). Calculus-based methods can be applied only to differentiable objective functions, can be unstable, and are susceptible to the *foothill problem* – getting trapped at a local optimum. The random and enumerative methods can be hopelessly inefficient when the search space is large.

A genetic algorithm (GA) is an alternative optimization procedure inspired by Darwinian evolution. A conventional GA uses a bit string which encodes the problem parameters. The GA keeps a population of ~100 such strings and applies genetic operators to them to generate new strings. Each new string is evaluated, and better strings are preferentially retained for the next generation.

GAs are more generally applicable and robust than calculus-based methods, and can be much more efficient than random search and enumerative methods. GAs have been successfully applied to pipeline control [5], structural optimization [6], recursive adaptive filter design [7], VLSI circuit layout [8], and many other problems.

GAs were developed by John Holland and his colleagues at the University of Michigan [9]. A short description of GAs appears in the next two subsections. A more extensive discussion may be found in reference [10].

4.1 DESCRIPTION

The genetic algorithms in this work follow this reproductive plan:

GENETIC ALGORITHM.

    (i)   *Create a random population of size N.*

    (ii)   *Sort the population in ascending order of fitness.*

    (iii)   *Repeat until there are $N + m$ members in all:*

        (a) *Choose randomly a member C_i, where $1 \leq i < N$ (uniform distribution). C_i is almost unbiased.*

11

(b) Choose randomly a member C_j, where $i < j \leq N$ (uniform distribution). C_j is biased toward more fit members.

(c) Choose randomly a genetic operator g_k (see below).

(d) Create a new member $C \leftarrow g_k(C_i, C_j)$, or $C \leftarrow g_k(C_j)$ if g_k uses only one member. Discard C if it is the same as either parent.

(e) Determine fitness of C.

(iv) *Merge the new members into the population, discarding duplicates.*

(v) *Remove all but the last (strongest) N members of the population.*

(vi) *Terminate or go to step (iii).*

m was set to $.20N$ to reduce the sorting effort without materially delaying entry of new members into the active population. However, in a typical application, determining the fitness of new members dominates the computational effort.

The most important genetic operator is *crossover*, shown in figure 6. This operator chooses two positions along the bit string and uses the enclosed fragment from one parent and the end fragments from the other parent. Less important is *mutation*, shown in figure 7, which reverses one or more bits chosen at random from the single parent.

The probabilities with which different genetic operators are chosen are themselves adjusted during the computation. Details are discussed in appendix B.

4.2 EXAMPLE

Suppose the problem is to find the value of x which maximizes

$$f_{12}(x) = e^{-4\ln 2\left(\frac{x-0.0667}{0.8}\right)^2} \sin^6(5.1\pi x + 0.5)$$

for the range $0 \leq x \leq 1$, a problem suggested by Goldberg and Deb [11].

In this case, the bit string has a single field of 16 bits. Let these bits be the Grav code for the integer i. (The Gray code is used instead of the usual binary code so that codes for neighboring values will differ by only one bit.) x is then simply $i/65536$.

Two crossing points are
selected at random

| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |

| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |

| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
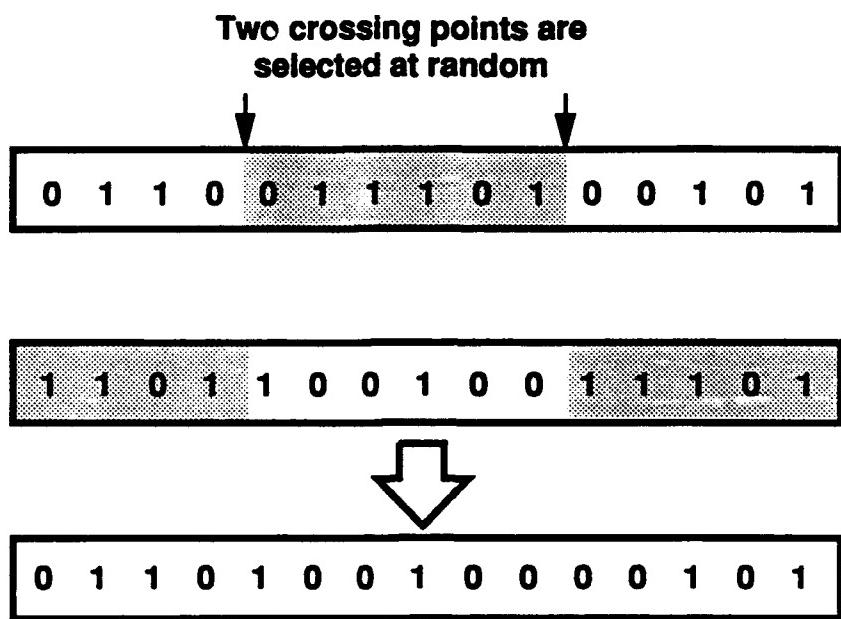
Figure 6. Crossover Operator

A fraction p of the bits are
selected at random to be flipped

| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |

| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
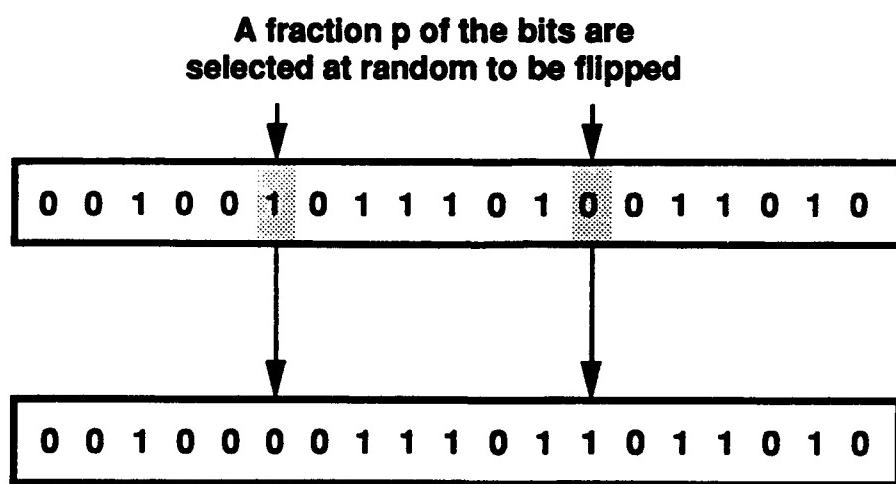
Figure 7. Mutation Operator

13
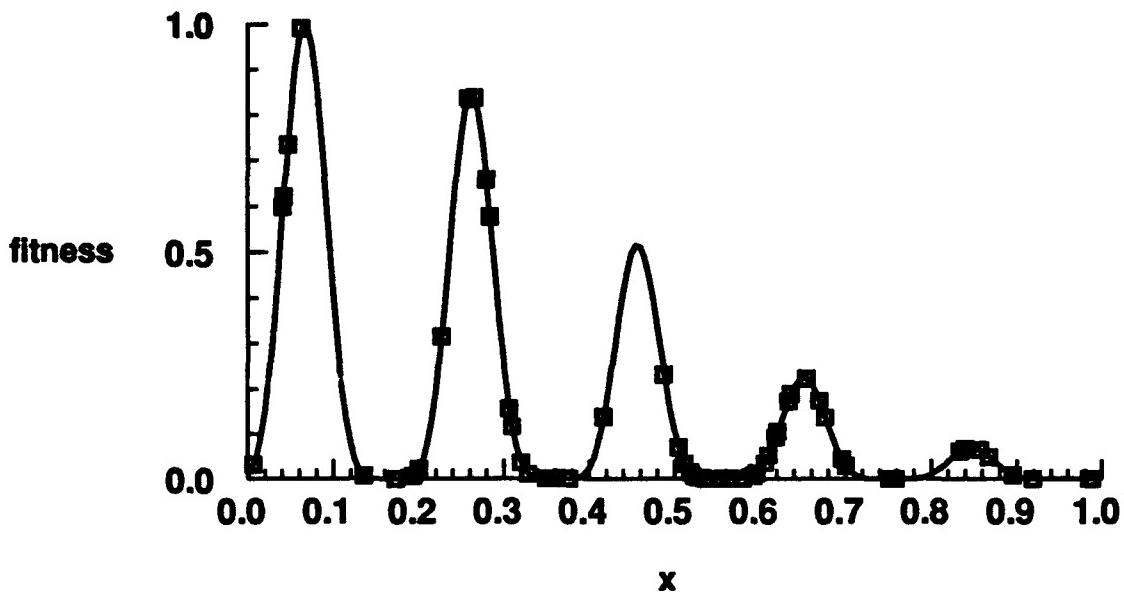
Figure 8. Initial Population

Crossover and mutation are the only genetic operators used.

An initial population of 60 is shown in figure 8. By the 10th generation, shown in figure 9, members are clustering around the local peaks of the fitness function. By the 30th generation, shown in figure 10, all members are clustered around the highest peak.

4.3 APPLICATION TO ROUTING

For the routing problem, a 4-bit field is used to select among 16 routing patterns that governed the heuristic solution of the traveling salesman problem. The patterns comprise the Sierpinski curve, the Hilbert curve, and 14 versions of the strips heuristic with different numbers and orientations of strips.

The remainder of the bit string consists of one bit per potential site, which is set if the site is to be visited and zero otherwise.

The "raw" fitness function is the sum of the values of the visited sites. The constraint on flight path length is handled by adding a *penalty function* as suggested by Goldberg [10]. Members satisfying the constraint are said to be *feasible*. Details are discussed in appendix C.

14
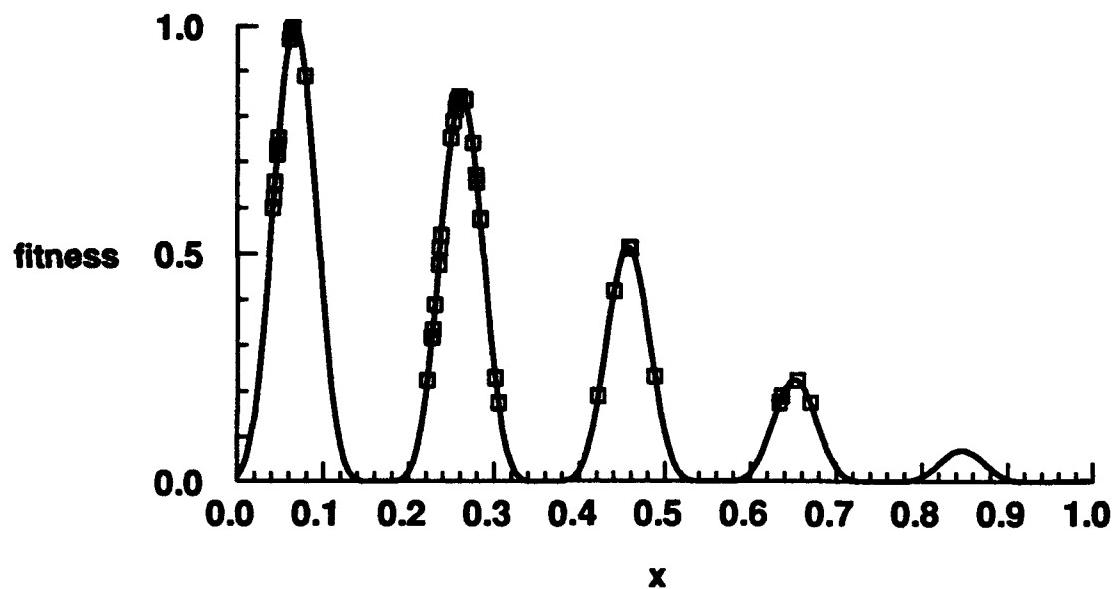
Figure 9. 10th Generation
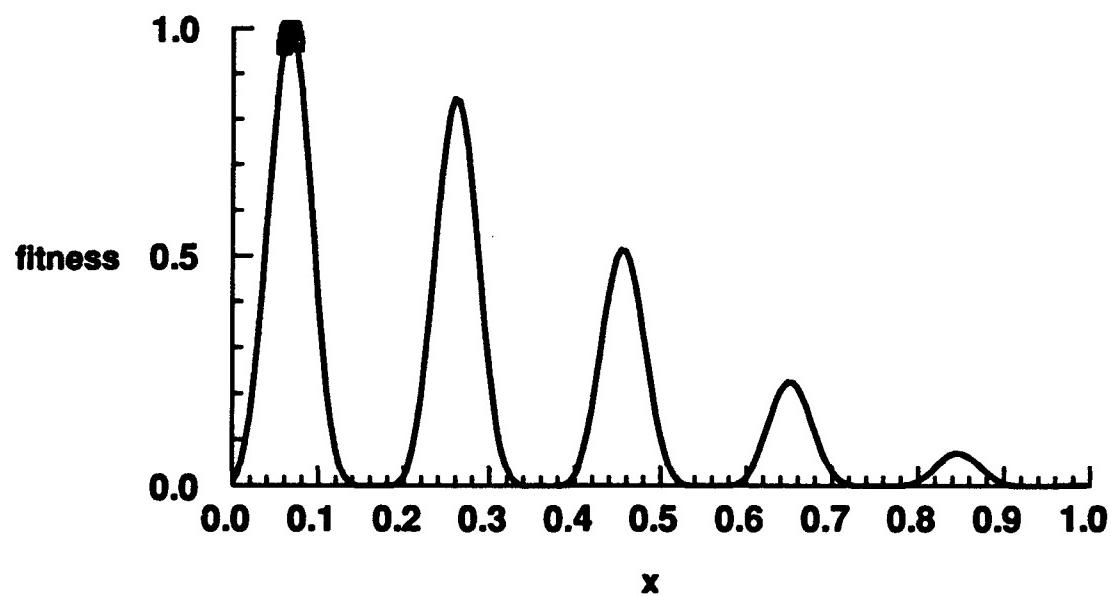
Figure 10. 30th Generation

15

Two genetic operators are used in addition to the crossover and mutation operators described above.

The *mutate-routing* operator, shown in figure 11, replaces the routing pattern with another chosen at random. The primary motivation for this is that changing the routing pattern and changing the visited sites are significantly different operations, perhaps best applied at different rates. Defining separate mutation operators allows the operator probabilities to adapt independently (see appendix B). Early in the calculation we would expect that changes in the routing pattern might well reduce the search route length, so that the probability of selecting the *mutate-routing* operator should be moderate. Late in the calculation we would expect that for a given member, the routing pattern and the set of sites to be visited would be tuned to one another. A change in routing pattern would be unlikely to decrease the search route length. However, site changes might still improve the search (by removing a site to shorten the path length or adding a site to increase the weight). Therefore it would be reasonable to mutate sites more often than routing patterns.

**One field is filled
with random bits**

0 1 1 0 | 0 1 1 1 0 1 0 0 1 0 0 0 1 1 0 0 0

1 0 1 0 | 0 1 1 1 0 1 0 0 1 0 0 0 1 1 0 0 0
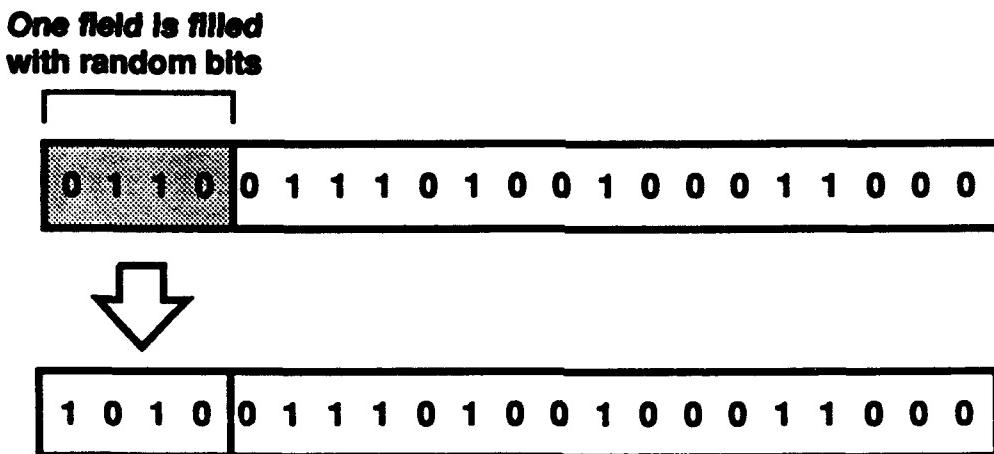
Figure 11. Mutate-Routing Operator

16

The other operator was motivated by the observation that the algorithm frequently chose routes that visited relatively low weight sites while skipping nearby sites with higher weights. This appeared to be due to the turn radius limitation. Merely adding the higher weight site may lead to the addition of a loop, whose length makes the path infeasible. On the other hand, deleting the low weight site reduces the value of the path. Thus moving the visit to the higher weight site would require two mutations (deleting one site and adding the other) but the intermediate states have low value and would be unlikely to survive. In GA terminology, the problem is *deceptive*.

The deception is overcome by combining the two mutations to form the *move-bit* operator. This operator, shown in figure 12, chooses a visited site at random and clears the corresponding bit in the bit string. It then finds the preceding and following set bit in the string and sets some different bit in the interval between them. This procedure makes sense only because the sites are rearranged in a preprocessing step so that neighboring bits represent nearby sites. (Actually, the sites are put into Sierpinski curve order.)
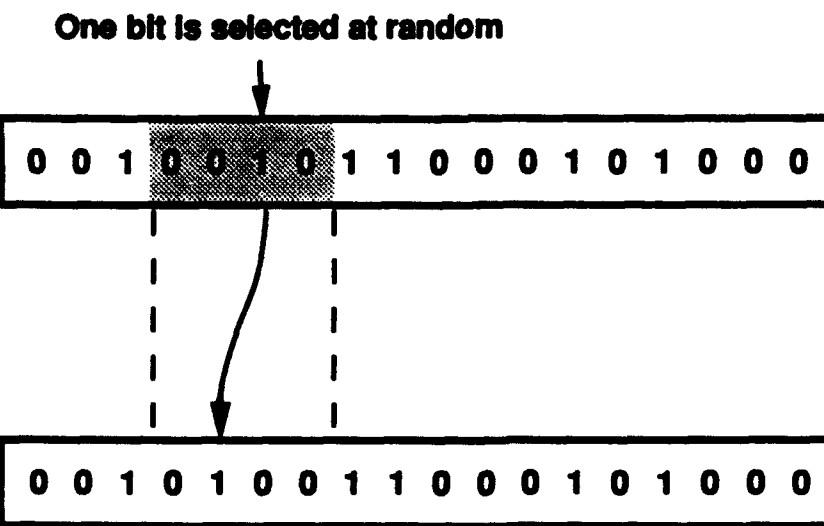
Figure 12. Move-Bit Operator

17
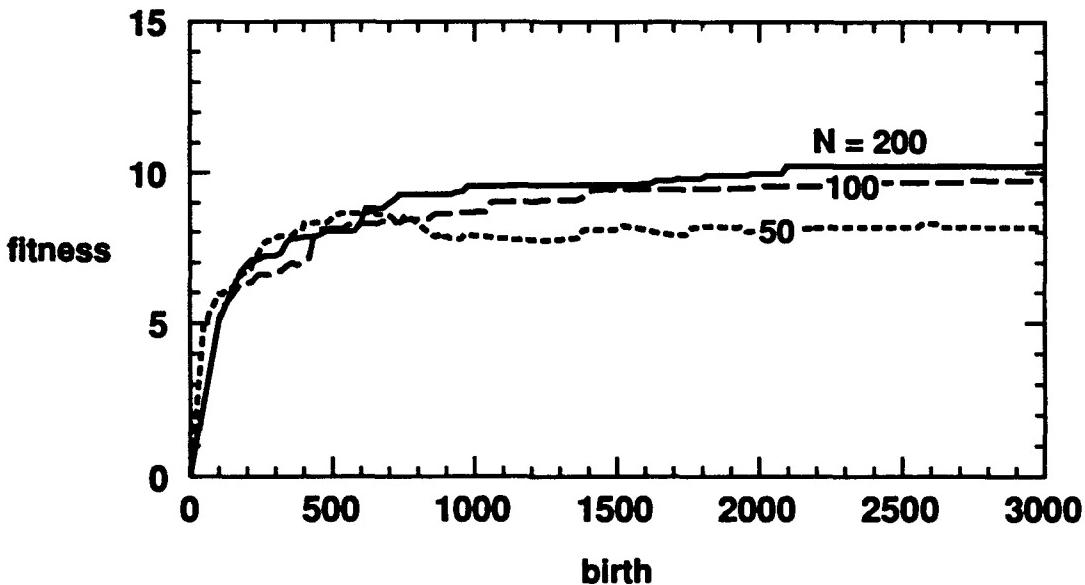
Figure 13. Score of Best Member

The progress of the calculation may be shown by plotting the raw fitness (sum of values of visited sites) of the best feasible member in each generation. *Typical plots are shown* in figure 13. The fitness increases rapidly at first, then more slowly. (The algorithm makes about 80 percent of its progress toward the final value in the first 20 percent of the time – a value which doesn't change much with different run times.)

One of the important tuning parameters is the population size. Populations of up to several hundred members were used. Generally, large populations take longer to converge but reach better solutions, as shown in figure 13. However, results are also poorer for very large populations when the total number of evaluations is held fixed. This is due to the tradeoff between population variability (i.e., presence of the building blocks for good solutions) and the number of generations.

If the population is too small, the members won't adequately span the search space. If the population is too large, then (for a fixed number of evaluations) not enough generations will pass to allow the best features of the various members to be combined. This is shown by the results for a 20 site problem shown in figure 14. Four runs were made for each population size, with each limited to 4000 evaluations. The figure shows the mean score and a one standard deviation range. The optimum population in this case is apparently near 150 members. The results for a 50 site problem shown in figure 15 suggest that the optimum population may increase for larger problems.

18
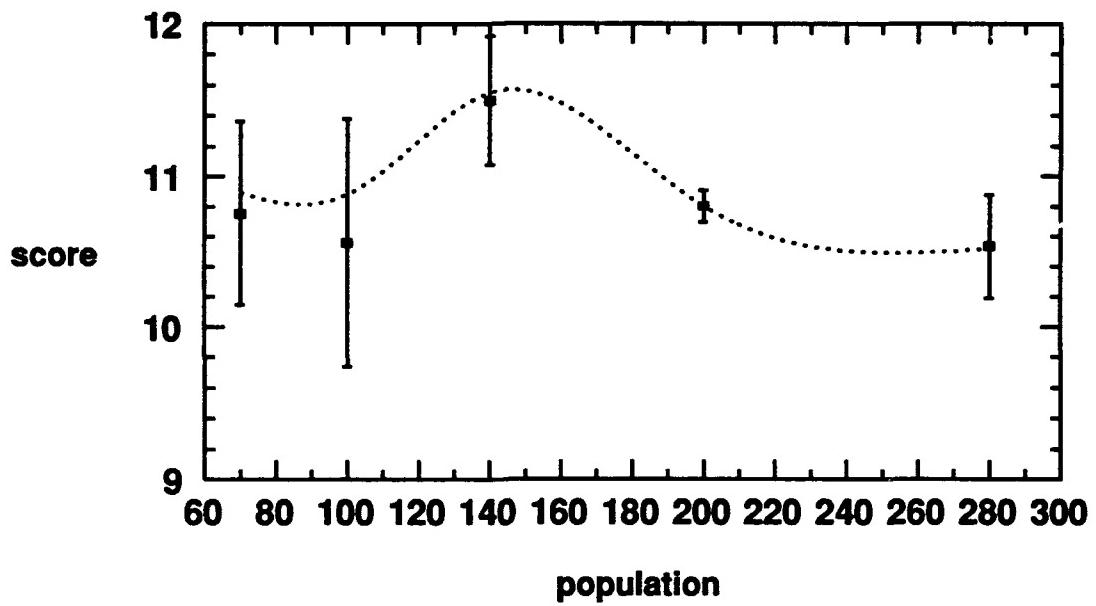
Figure 14. Scores as a Function of Population for 20 Site Problem
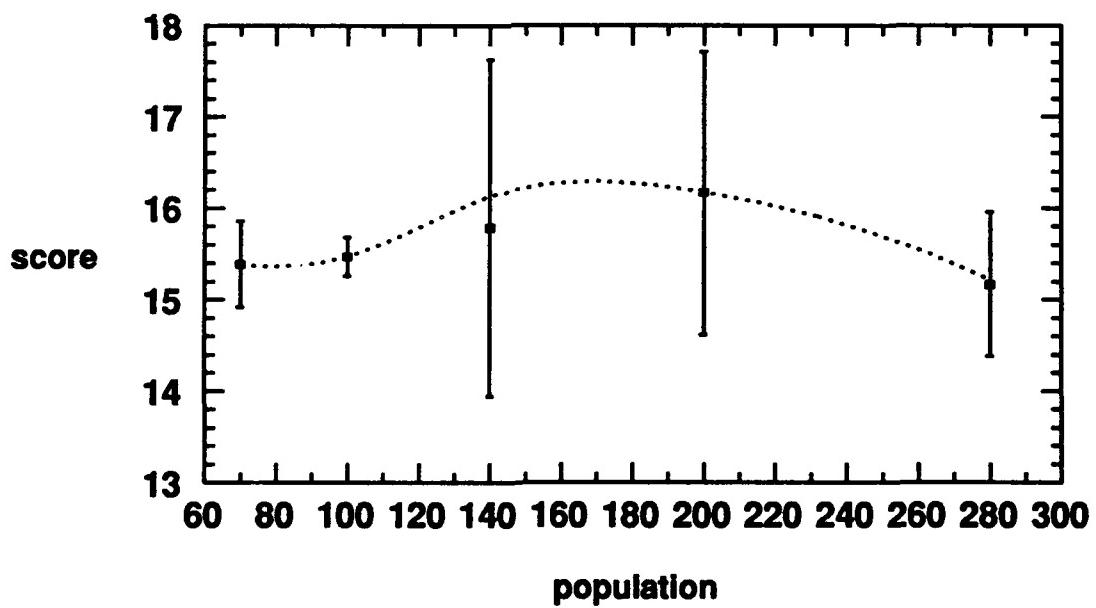
Figure 15. Scores as a Function of Population for 50 Site Problem

19

SECTION 5

RESULTS

Figure 16 shows a sample problem with 20 sites. The path is required to start at the point labeled 1 and end at the point labeled 2. The other sites are shown as circles with areas proportional to their weight (that is, the expected likelihood of finding the target there).
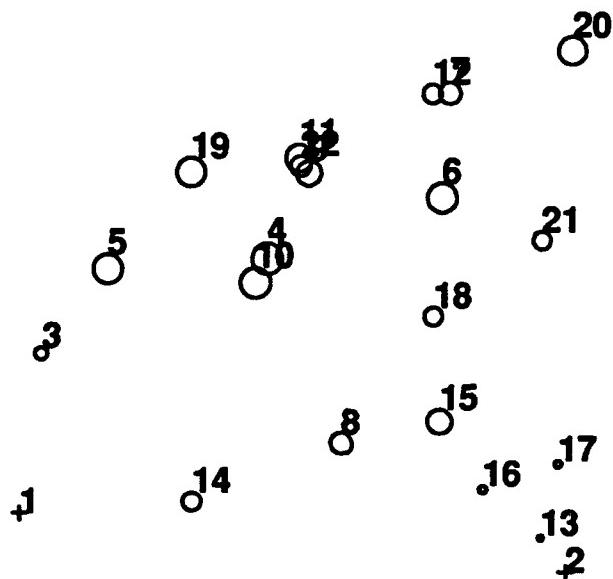
Figure 16. Sample Problem with 20 Sites

Figure 17 shows the best path found through those sites with path length restricted to 3.

Figure 18 shows a second sample problem with 50 sites. The variation among paths found by the genetic algorithm can be judged from figures 19–21, which show the three best paths with length less than 5.
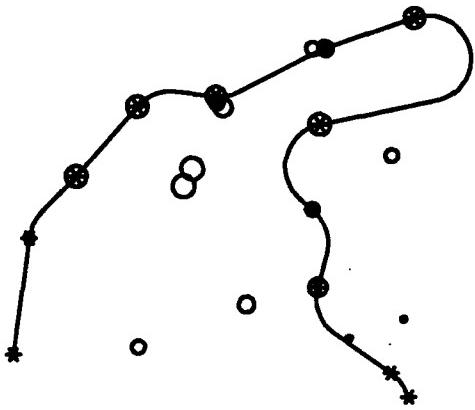
Figure 17. Sample Path Through 12 of 20 Sites

None of the above runs used the arbitrary insertion routing procedure, relying instead on spacefilling curves and strips routing patterns. Detailed results and genetic algorithm parameters are listed in table 1.
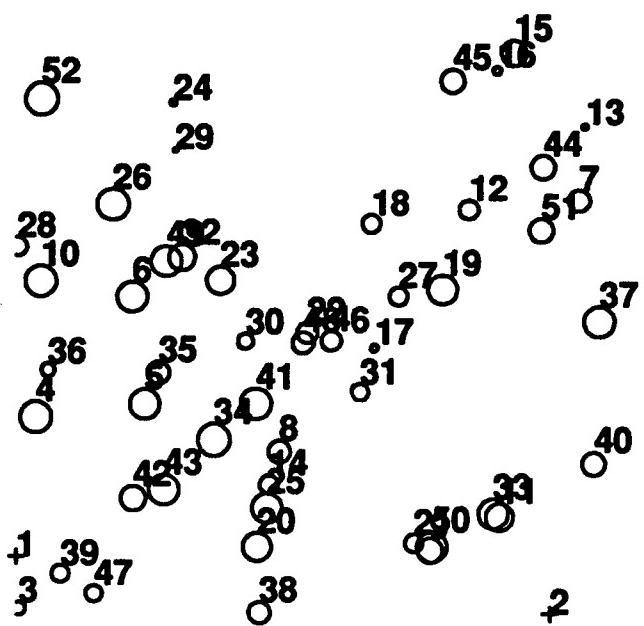
Figure 18. Sample Problem with 50 Sites
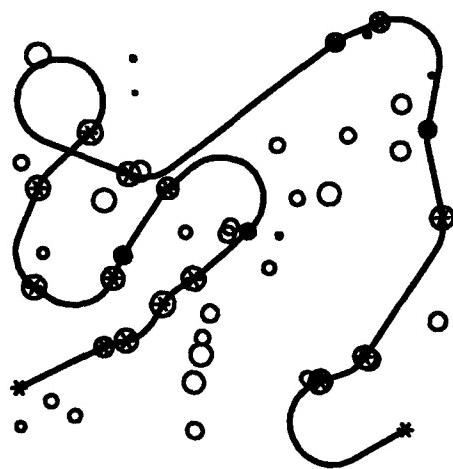
Figure 19. Path Through 20 of 50 Sites (Length 4.91, Weight 27.23)
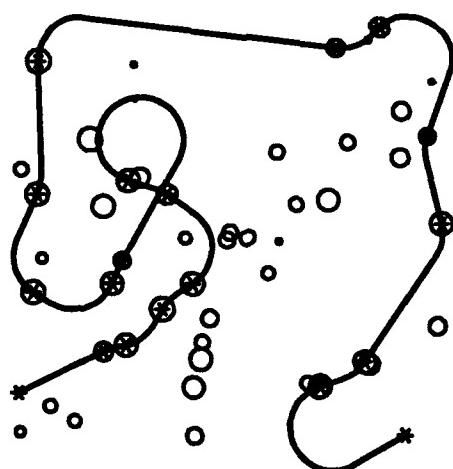
Figure 20. Path Through 19 of 50 Sites (Length 4.94, Weight 26.55)
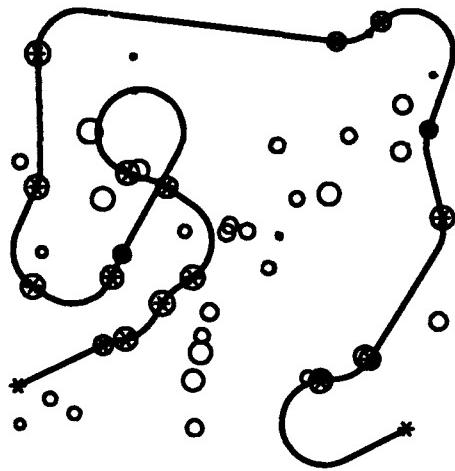
Figure 21. Path Through 19 of 50 Sites (Length 4.99, Weight 26.31)

Table 1. Parameters for Sample Results

Cities	Visited	Weight	Length	Population	Evaluations	Figure
20	12	11.14	2.78	120	4000	17
50	20	27.23	4.91	200	3000	19
50	19	26.56	4.94	200	3000	20
50	19	26.31	4.99	200	3000	21

SECTION 6
DISCUSSION

A previous report presented a fast heuristic based on spacefilling curves for the covering salesman problem. It finds a two-dimensional route which passes within a specified distance (corresponding to sensor range) of all the sites of interest, while approximately minimizing the total distance traveled. This report extends that procedure to include the selection of an appropriate subset of sites when their number is large, so that time and fuel constraints preclude visiting all of them. The finite turn radius is also taken into account.

This work disregards sensor range, but only for simplicity. A previous report described procedures for eliminating some visits because those cities are within sensor range of cities which are visited [2]. Those procedures could be applied with no significant changes.

A bigger effort would be needed to permit revisits of sites, and to properly accumulate credit for multiple visits. Neither the spacefilling curve heuristic nor the strips heuristic easily allows multiple visits. The arbitrary insertion heuristic could accommodate a starting route with multiple visits, and could be adapted to generate such a route as well.

This work still disregards defenses, terrain avoidance, area searches, waypoint offset (so the aircraft does not pass directly over a site), and sensor pointing.

Much of the computational effort of this procedure is in accounting for the finite turn radius of the aircraft. It might be improved by first calculating a lower bound (the length of the piecewise straight line path) and discarding the path if it is far from feasible.

LIST OF REFERENCES

1. Current, J. R., and D. A. Schilling, 1989, "The Covering Salesman Problem," *Transportation Science*, Vol. 23, No. 3, pp. 208–213.

2. Van Zandt, J. R., 1990, *Spacefilling Heuristics for Search Routes*, ESD-TR-90-341, AD B156315, Electronic Systems Division, AFSC, Hanscom Air Force Base, Massachusetts.

3. Karp, R. M., and J. M. Steele, 1985, "Probabilistic Analysis of Heuristics," in *The Traveling Salesman Problem* (E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds.), New York, NY: John Wiley & Sons, pp. 184, 194.

4. Golden, B. L., 1985, "Empirical Analysis of Heuristics," in *The Traveling Salesman Problem* (E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds.), New York, NY: John Wiley & Sons, pp. 217, 223.

5. Goldberg, D. E., 1983, "Computer-aided gas pipeline operation using genetic algorithm and rule learning," (Doctoral dissertation, University of Michigan), *Dissertation Abstracts International*, Vol. 44, No. 10.

6. Goldberg, D. E., & M. P. Samtani, 1986, "Engineering optimization via genetic algorithm," *Proceedings of the Ninth Conference on Electronic Computation*, pp. 471–482.

7. Etter, D. M., M. J. Hicks, & K. H. Cho, 1982, "Recursive adaptive filter design using an adaptive genetic algorithm," *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, Vol. 2, pp. 635–638.

8. Davis, L., & D. Smith, 1985, *Adaptive design for layout synthesis*, internal report, Texas Instruments.

9. Holland, J. H., 1975, *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: The University of Michigan Press.

10. Goldberg, D. E., 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, Mass.: Addison-Wesley.

11. Goldberg, D. E., & K. Deb, 1987, "Genetic Algorithms with Sharing for Multimodal Function Optimization," *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, Morgan Kaufmann Publishers, Inc., Palo Alto, CA.

12. DeJong, K., 1975, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, University Microfilms No. 76-9381, University of Michigan.

13. Grefenstette, J. J., 1986, "Optimization of Control Parameters for Genetic Algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-16, No. 1, pp. 122–128.

14. Davis, L., 1989, "Adapting Operator Probabilities In Genetic Algorithms," *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, pp. 61–69, Morgan Kaufmann Publishers, Inc., Palo Alto, CA.

15. Richardson, J. T. et al., 1989, "Some Guidelines for Genetic Algorithms with Penalty Functions," *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, pp. 191–197, Morgan Kaufmann Publishers, Inc., Palo Alto, CA.

16. Siedlecki, W. & J. Sklansky, 1989, "Constrained Genetic Optimization via Dynamic Reward-Penalty Balancing and Its Use in Pattern Recognition," *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, pp. 141–150, Morgan Kaufmann Publishers, Inc., Palo Alto, CA.

APPENDIX A
FLIGHT PATH ALGORITHM

A.1   BACKGROUND

Certain properties of complex arithmetic can be used to construct a flight path consisting of straight lines and circular arcs.

Of course, addition of complex numbers corresponds directly to addition of two-dimensional vectors. Multiplying a complex number by a scalar (a real number) corresponds to multiplying a vector by a scalar, and results in a vector with the same direction but in general a different length.

Multiplying a complex number by a second complex number corresponds to a rotation together with scaling. For example, multiplying by i corresponds to a counterclockwise rotation by $90°$ about the origin. Multiplying by $e^{i\theta}$ corresponds to a counterclockwise rotation by an angle θ about the origin, where θ is in radians.

These concepts can be used to construct equations expressing the desired geometric relationships, which can then be solved for the unknown values. This is often simpler than using analytic geometry alone.

A.2   WAYPOINT TURN ALGORITHM

Let $s_1 \ldots s_n$ be the points to be visited. (In the following, complex values are denoted by lower case letters, while upper case letters denote real values.) We assume that no three points to be visited are along a straight line, so that each point will correspond to a turn. For now, we will assume each turn has the same radius R. A sample flight path is shown in figure 22. It consists of a straight leg from b_k to the "turn point" e_k, a circular arc centered at c_k which passes through the point s_k, and a second straight leg from b_{k+1} to e_{k+1}.

In the waypoint turn algorithm, the turn point coincides with the visited point:

$$e_k = s_k. \qquad\qquad (A-1)$$
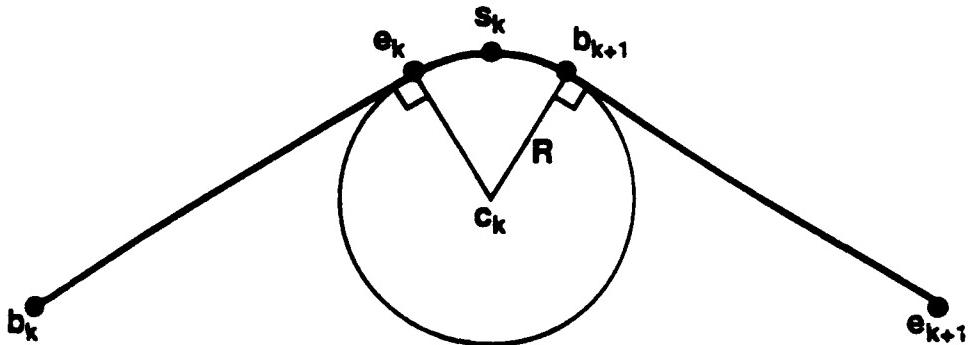
Figure 22. Flight Path for Single Turn

This means the turn center c_k is at a distance R from e_k along a line perpendicular to the preceding straight segment. This relationship corresponds to the equation

$$c_k = e_k + (e_k - b_k)\frac{R}{L_k}(\mp i), \qquad (A-2)$$

where the upper sign corresponds to a right turn and $L_k = |e_k - b_k|$ is the length of the kth straight segment.

The turn direction can be deduced from the direction to the next site. Specifically, we expect a right turn if

$$\text{Im}\left(\frac{s_{k+1} - s_k}{s_k - b_k}\right) < 0. \qquad (A-3)$$

The length L_{k+1} of the next straight segment can be found using the Pythagorean theorem,

$$L_{k+1}^2 = |s_{k+1} - c_k|^2 - R^2. \qquad (A-4)$$

If the right hand side of (A-4) is negative, the next visited point is inside the turn circle so that the flight path is infeasible. In this case it is only necessary to reverse the direction of the turn at e_k and recalculate c_k. This adds a *loop*.

32

The equation for c_k in terms of the following straight segment is

$$c_k = b_{k+1} + (e_{k+1} - b_{k+1}) \frac{R}{L_{k+1}} (\mp i). \qquad (A-5)$$

Solving this for b_{k+1}, we have

$$b_{k+1} = \frac{c_k \pm i\, e_{k+1} R/L_{k+1}}{1 \pm i\, R/L_{k+1}}. \qquad (A-6)$$

A.3 REMARKS

The turn radius R has been assumed constant, but allowing a different radius at every turn would introduce no significant difficulties.

These formulas can be used to generate flight paths through specified points consisting of straight lines and circular arcs. To use the method, the centers of the turn circles must be found. A simple method for determining these centers, under the assumption that the turn points coincide with the points to be visited, has been implemented. A more elaborate method of positioning these centers could generate shorter flight paths. However, the difference would be significant only for acute turns. If the preceding and following turn points are distant, the difference for a turn of θ is

$$\delta L = R\left(\sin\left(\theta/2\right) - \sin\theta\right) = 2R\sin\left(\theta/2\right)\left(1 - \cos\left(\theta/2\right)\right) = O(\theta^3) \qquad (A-7)$$

This amounts to $2R$ for a turn of 180°, but $.414R$ for a turn of 90° and only $.058R$ for a turn of 45°. Note that a good choice of visitation order (i.e., a good solution to the traveling salesman problem) will tend to give small turn angles.

APPENDIX B
ADAPTIVE OPERATOR PROBABILITIES

The efficiency of a genetic algorithm depends on the settings of several parameters including population size and the probabilities of applying each genetic operator. De-Jong found robust settings by using a test suite of function optimization problems [12]. Later, Greffenstette used a genetic algorithm to find a better set of parameter values [13]. However, these studies used only the crossover and mutation operators.

When additional operators are used, it is necessary to set their selection probabilities. Setting them by either of the above methods would require a very considerable effort. It is attractive to set them adaptively instead.

The following scheme for adaptively setting operator probabilities was presented by Davis [14]. Each new individual that is better than the current best individual acquires a "local delta" equal to the improvement. At intervals of I evaluations, each of the last W individuals pass back to its parent(s) P times the sum of its local delta and the deltas inherited from its progeny (except that no delta is passed back more than M generations). Each individual then passes its remaining delta to the operator which created it. The operators accumulate the deltas in a vector C. The vector V of operator probabilities is updated by shifting a fraction S, so that $V \leftarrow SC/|C| + (1 - S)V$.

The five parameters W, I, S, P, and M effectively replace the operator probabilities as governing parameters. Davis found the values $W = 100$, $I = 50$, $S = 0.15$, $P = 0.15$, and $M = 10$ to be effective.

There are two points that seem to make implementation of this awkward. The first is that even deleted members would have to be kept around, on the chance that one of their progeny would pass back some delta. The other concerns the handling of M, the limit on the number of generations that delta is passed back. If M could be disregarded, the deltas could be propagated in one pass, starting with the most recent member, with each step involving only the member's immediate parents. Accounting for M would seem to require a walk of the family tree of each member having a local delta, to a depth of M or the edge of the adaptation window (whichever came first). This procedure would seem to grow exponentially with W/population, and would be particularly lengthy if two-parent operators were successful (leading to large family trees).

In addition, one might want to reward operators when they created individuals that were better than any of their parents, regardless of whether they were better than the

previous best individual. This would give the adaptation mechanism more information to work with.

Because of the above concerns, the following somewhat simpler scheme has been used instead.

Each member has an associated "responsibility vector" R which indicates the relative roll each genetic operator had in its creation. If operator g_k is used to create child c from a single parent p, the new member gets a responsibility vector

$$R_c = PR_p + (1 - P)U_k, \tag{B-1}$$

where U_k has a 1 in the kth position and 0s elsewhere. If the operator used two parents m and f, the child gets a responsibility vector

$$R_c = P(R_m + R_f)/2 + (1 - P)U_k. \tag{B-2}$$

The operators accumulate credit for improvements in a vector C. If the fitness of the new member exceeds that of its better parent by δ, C is incremented by $\delta R_c/|R_c|$. If the new member is less fit than at least one parent, C is not changed. Each generation (m births), the vector V of operator probabilities is updated by shifting a fraction S:

$$V \leftarrow SC/|C| + (1 - S)V, \tag{B-3}$$

and the credit vector is attenuated:

$$C \leftarrow \mu C. \tag{B-4}$$

For this work, $P = 0.90$, $S = 0.30m/100$, and $\mu = 0.50$.

This scheme follows the two principles suggested by Davis, namely that the probability of applying an operator should be altered in proportion to the performance of the individuals it creates, and that local measures of performance are insufficient (that is, operators creating ancestors of successful individuals should also be rewarded).

An example of operator probability adaptation is shown in figure 23. Each curve is an average over four runs. The adaptive procedure confirms the importance of the crossover operator, since its probability steadily increases. On the other hand, it suggests the "move-bit" operator may be relatively unhelpful (or perhaps helpful only early in the optimization process).
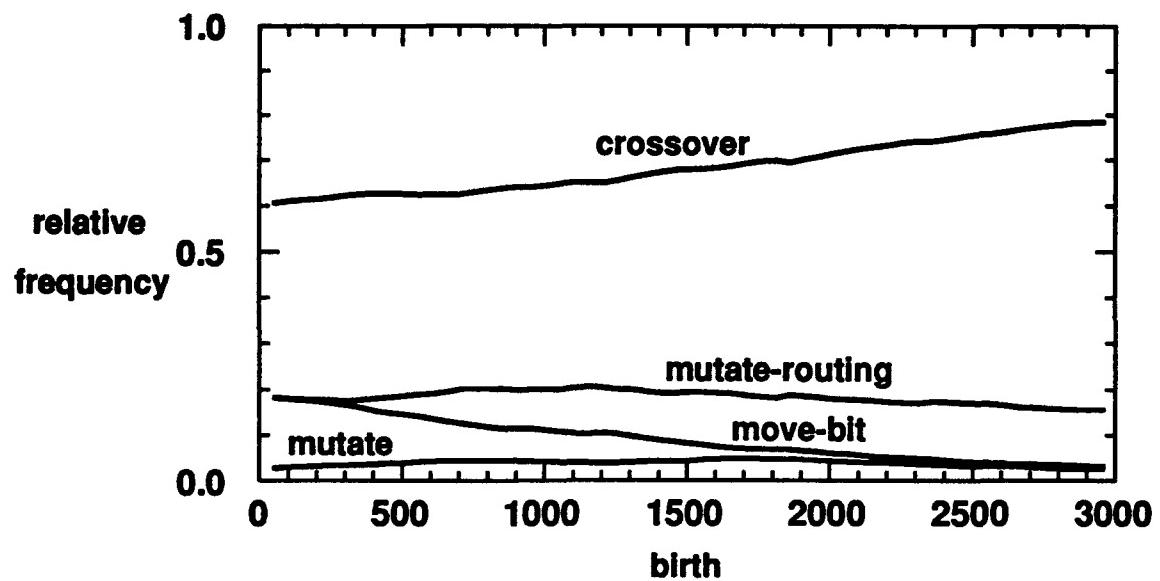
Figure 23. Operator Probability Adaptation

APPENDIX C
CONSTRAINED OPTIMIZATION

Suppose we wish to use a genetic algorithm to minimize the function

$$f(x) \qquad\qquad (C-1)$$

while satisfying the constraint

$$g(x) < 0. \qquad\qquad (C-2)$$

Goldberg suggests a penalty function approach so that the objective function is replaced by a compound objective function:

$$f_s(x) = f(x) + \alpha\Phi(z), \qquad\qquad (C-3)$$

where $z = g(x)$, α is a nonnegative *penalty coefficient* and Φ is a nonnegative *penalty function* [10].

Richardson *et al.* found that a penalty function should not be too harsh because the most direct path from a given solution to the optimal solution may require some infeasible intermediate structures [15]. Therefore, the penalty function should be chosen to balance the preservation of information against the pressure for feasibility.

It is necessary to penalize infeasible members, but not so much that moderately infeasible members cannot contribute information. I also want to reward moderately feasible members for satisfying the constraint well. The penalized fitness function should have a maximum at the *feasibility boundary* (where $g(x) = 0$). Thus, I wanted a penalty function which is smooth at 0 and has a slope there equal to f'/g'. On the other hand, I didn't want members well inside the feasibility boundary to be either penalized or rewarded, so the penalty function should be zero for large negative values. Members well outside the feasibility boundary should be heavily penalized, so the penalty function should be large for large positive values.

I chose the penalty function

$$\Phi(z, w) = ze^{z/w}, \qquad\qquad (C-4)$$

which is shown with a solid line in figure 24.

A simpler alternative which works almost as well is

$$\Phi(z,w) = \begin{cases} \dfrac{z}{\left(1 - \frac{z}{w} - \frac{z^3}{w^3}\right)}, & \text{for } z \leq 0 \\[2em] z\left(\dfrac{z}{w} + 1\right), & \text{for } z > 0 \, , \end{cases} \qquad (C-5)$$

which is shown with a dotted line in figure 24. Each function has a unit slope at $z = 0$, so the compound objective function will have a local minimum at the feasibility boundary if $\alpha = f'/g'$. α then plays the role of a Lagrange multiplier. Each function also has a second parameter w, a characteristic width, which is discussed below.
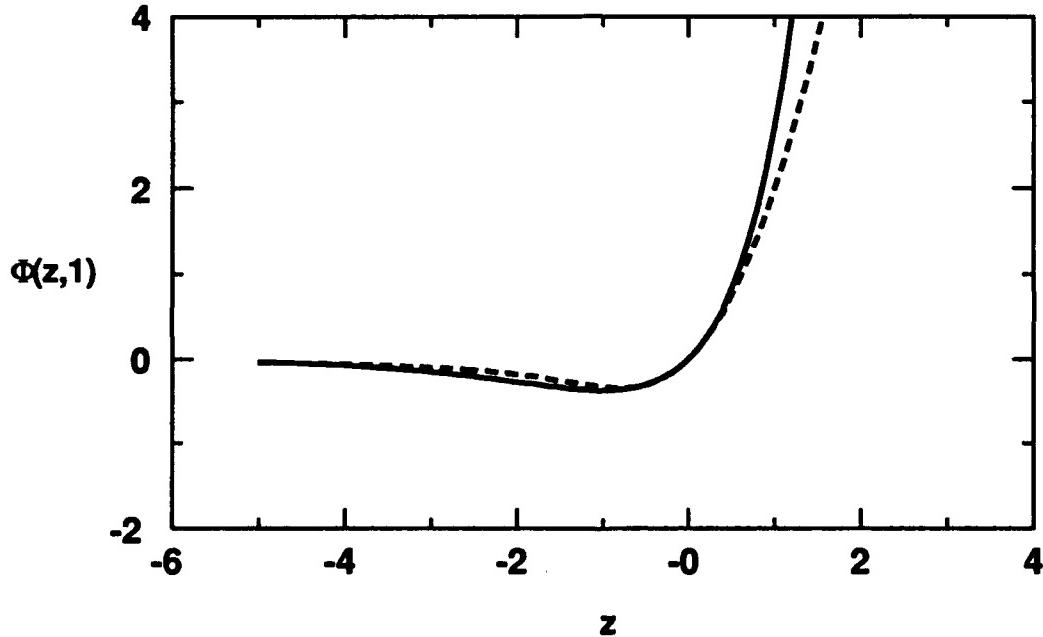

Figure 24. Penalty Functions

Siedlecki and Sklansky introduced the concept of *dominance* [16]. Recall that we have assumed a minimization problem, so that better solutions correspond to smaller values of f. Solution x_i *dominates* solution x_j if $f_s(x_i) < f_s(x_j)$ for any positive value of α. Evidently, this implies both

$$f(x_i) < f(x_j) \qquad (C-6)$$

and

$$g(x_i) < g(x_j). \qquad (C-7)$$

40

If α is too large the optimal solution may be precluded, and if α is too small then members will tend to be infeasible. It is therefore important that α be close to f'/g'. This ratio is estimated from the current population (specifically, from dominant members on either side of the feasibility boundary) using a single state Kalman filter as follows:

DERIVATIVE ESTIMATE.

(i)   *Sort the members on their constraint values.*

(ii) *Find a fit member near the feasibility border (the most fit member x_f with constraint $-1 < g(x_f) < 0$).*

(iii) *Find an infeasible member near the feasibility border (the most fit member x_i with constraint $0 < g(x_i) < 1$).*

(iv) *If the moderately infeasible member was more fit (that is, $f(x_i) < f(x_f)$), the derivative f'/g' is estimated as $\alpha' = (f(x_i) - f(x_f))/(g(x_i) - g(x_f))$. The filtered derivative estimate is updated according to $\alpha \leftarrow (P\alpha' + R\alpha)/(P+R)$, and its estimated covariance P is updated according to $P \leftarrow RP/(R + P) + Q$.*

(v)   *Otherwise, $P \leftarrow P + Q$.*

Q, R, and P are initialized in the ratio $1 : 10 : 10^{11}$, so the *Kalman filter gain* $P/(P+R)$ is very high on the first iteration, drops quickly thereafter, and doesn't rise again unless there are several generations in a row with no improvements.

For some runs, the width w was kept constant at 2. However, there was some evidence that decreasing w made the accurate determination of α less critical. This was done by initially setting w to 1, and thereafter to the range of the constraint function for the current population:

$$w \leftarrow z_{max} - z_{min}. \tag{$C-8$}$$